Towards Resilient Autonomous Navigation of Drones

Angel Santamaria-Navarro*, Rohan Thakker*, David Fan, Benjamin Morrell, and Ali-akbar Agha-mohammadi

angel.santamaria.navarro@jpl.nasa.gov
WWW home page: http://angelsantamaria.eu
NASA-Jet Propulsion Laboratory, California Institute of Technology
Oak Groove Dr. 4800 Oak Grove Dr, Pasadena, CA 91109, USA,

Abstract. Robots and particularly drones are especially useful in hazardous environments which are unreachable to humans. In these situations, a reliable and robust state estimation solution is critical as the environments can be very challenging for both perception and mobility, such as in underground, disaster zones and collapsed buildings. Recently, most of the research in robot state estimation has focused on developing complex algorithms favoring accuracy. However, practically all these algorithms make a big assumption: the main estimation engine will not fail during operation. In contrast, we propose an architecture that pursues robustness in state estimation by considering redundancy and heterogeneity in both sensing and estimation algorithms. The architecture is designed to detect failures and adapt the behavior of the system to ensure safety. We present HeRO (Heterogeneous Redundant Odometry), a stack of estimation algorithms running in parallel and supervised by a resiliency logic. This logic carries out three main functions: a) perform confidence tests both in data quality and algorithm health; b) re-initialize those algorithms that might be malfunctioning; and c) ensure continuity in the output while switching between estimation sources. The output of the resiliency logic is fed directly to guidance and control modules of the robot. The validation and utility of the approach is shown with real experiments on a flying robot for the use case of autonomous exploration of subterranean terrains, with particular results from the STIX event of the DARPA Subterranean Challenge.

1 Introduction

Robots are especially useful for accomplishing tasks that would otherwise be impossible for humans to do. From search and rescue missions in collapsed mines, to package delivery for disaster response or commercial applications, and even to exploration of other planetary bodies, robots have the potential to push

*Both authors contributed equally to this manuscript

the boundaries of human capability. However, much research in robotics focuses on deploying robots in controlled, laboratory-like settings which are friendly to humans. Deploying robots in real-world settings demands autonomy which is *robust* and *resilient* to different types of failures. Such failures include loss of communications with human operators, physical damage to sensors, loss of perceptual data in degraded environments (*e.g.*, dust, smoke or fog), and loss of GPS/GNSS. These failures can often result in damage to the robot, injuries to humans and failure of mission-critical tasks.

In this work we focus on a particular class of failures related to state estimation. In the last few years, approaches tackling localization and state estimation of robots (*e.g.*, estimating position, orientation or velocity) have shown sufficient maturity for using them as sensing feedback in robot controls, enabling autonomous operations. However, failure in state estimation is often catastrophic, cascading down through the whole autonomy stack. For example, poor position estimates can result in inaccuracies in mapping and planning, while poor velocity estimates can result in dangerous and unstable control. These impacts are of particular importance on aerial robots, where autonomous capabilities are always limited by the state estimation quality.

Most state estimation methods are developed with a focus on the accuracy of the estimations. For example, there is a comparison of more than 100 different approaches in the visual odometry / SLAM evaluation of the KITTI dataset [1] The most accurate methods usually fuse data from different sensors in a tightly coupled fashion, with a single estimation engine (usually filtering or with iterative optimization procedures). For example, fusing visual information (monocular, stereo or RGB-D camera configurations), range measurements and inertial measurements (IMU)[2,3,4,5,6,7] In these approaches, a high precision outcome is attained by jointly estimating a subset of past sensor poses and a number of landmarks in the environment, creating a graph of constraints between robot states and landmarks, which is incrementally solved. These solutions are computationally intensive and highly dependant on good outlier rejection techniques. In case of unsuccessful rejection of invalid measurements (happening in most of real situations), the resulting estimation accumulates drift or even fails, producing catastrophic consequences on the controls. Moreover, relying on a single tightly coupled approach to control the platform disables the option of an online re-initialization in case of estimation failures. In contrast to tightly coupled, one can use loosely coupled approaches where the state estimation is provided by separate algorithms (dedicated to smaller number of sensors) and merged afterwards, producing partial or complete state estimations. For example, in [8] fuse measurements from an optical flow sensor that directly provides Cartesian velocities with a vertical ranger and an IMU; or [9,10] which use relative pose estimates from visual information as a prior for a LiDAR odometry estimation to improve the accuracy. .

In order to address the fragility of state estimation methods in general, we propose a generic framework for robots which tracks and adapts to these types of failures. Inspired by lessons learned from the development of robotics

in space exploration, we advocate an architecture for resilient robots which relies on the notions of redundancy, identifying and characterizing failures, and adapting to those failures (see Figure 1). This "resiliency architecture" can be divided into three sub-systems: 1) Hardware 2) State Estimation and 3) Guidance and Control. Resiliency in hardware means mechanical protections and redundancy/heterogeneity in both sensors and actuators. Resiliency in state estimation is the main focus of this work and to this end we propose a novel state estimation algorithm which intelligently combines heterogeneous, redundant odometry algorithms (HeRO). Resiliency in guidance and control entails guaranteeing appropriate plans and actions in accordance with the level of available information, especially with respect to the quality and status of the state estimation.

The main idea behind our state estimation approach HeRO is to create a resiliency logic which enables the use of several state estimation methods in a loosely coupled solution. We run multiple odometry approaches in parallel and supervise them with this resiliency logic, which can check both sensor data integrity and the health of the algorithms. Methods which do not pass confidence tests are required to reinitialize and, if this entails a switch in the estimation source used for controls, the resiliency logic takes care of continuity in the produced output. Hence, the resiliency logic is fed with all different estimation modalities, chooses the best source for the controls and re-initializes those methods considered to be failed. To keep global track we maintain a separate global localization module that is not required to run at a high frequency and is not used to control the robot, but rather only updates global positioning. The main advantages of the proposed scheme are the following:

- Allows hardware and software redundancy.
- Distributes among all algorithms the responsibility of overall success.
- Allows switching between methods and/or re-initializing a complete algorithm.
- Enables parallel processing. Each individual estimator can run in its own process or even hardware, and we take advantage of the resulting estimated states.
- Minimizes the effect of a particular sensor failure. We can easily use methods combining sensors based on different physical phenomena, minimizing those zones where no estimation can be produced.
- Allows fast integration of new sensor modalities and methods. Methods can be tested and tuned independently.

The main contributions of this paper are four-fold. First, we present a general "resiliency architecture" for robust autonomous robots which leverages the concepts of redundancy, identifying failures, and adapting to those failures. Next, we present HeRO, a novel state estimation scheme which uses confidence tests on raw data and state estimation quality to detect incorrect sensor measurements or algorithm failures. Furthermore, we demonstrate how HeRO incorporates a multiplexing logic that enables live switching between estimation methods and re-initialization of algorithms or sensors that failed. Finally, we show results of

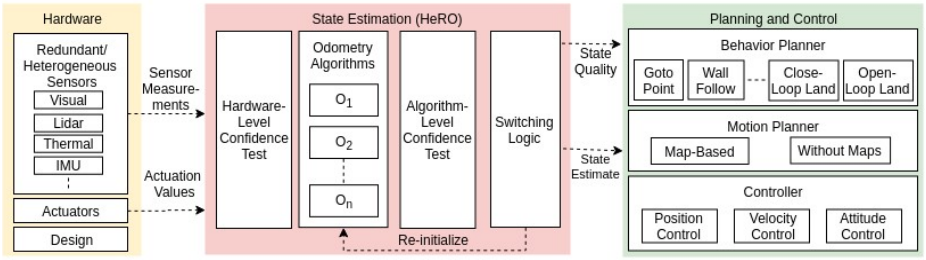


Fig. 1: General resiliency architecture.

experiments with a real robot to validate HeRO as well as the general resiliency architecture. More details of these experiments are shown in the accompanying video.

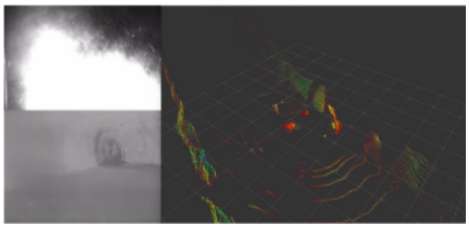
The remainder of this article is structured as follows. In the following section 2 we describe the general resiliency architecture, with corresponding concepts and solutions. Validation and experimental results are presented in Section 3, showing the feasibility of the proposed approach through real robot experiments performed live at the STIX event in the DARPA subterranean challenge. Finally, conclusions are given in Section 4.

2 Resiliency Architecture

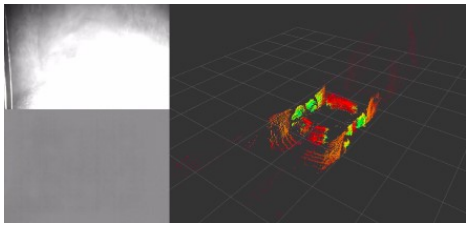
We propose a resiliency architecture that not only takes failures into account, but expects failures to occur. The architecture is designed so that the system can autonomously detect and adapt to failures. The key concept behind this architecture is to switch between heterogeneous and redundant state estimation streams based on confidence checks that assess the quality of those streams. The behaviour of the robot is also adapted by selecting the appropriate planner and controller given what reliable state estimation information is available. Although this work focuses on drone applications, the framework can be adapted to any vehicle. This general architecture is depicted in Figure 1 and is described hereafter.

2.1 Hardware

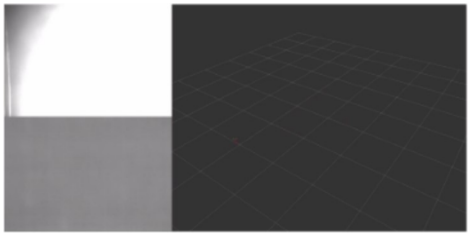
The robot hardware consists of a set of heterogeneous and redundant sensors that are used by the state estimation sub-system (HeRO) to run multiple odometry algorithms such as LiDAR-Inertial-Odometry (LIO), Visual-Inertial-Odometry (VIO), Optical Flow, etc. These sensors can be a combination of, for example, visible/infrared /thermal cameras, LiDAR (scanners or height sensors), IMU, Radar, and sonar. A variety of sensors are used to diversify the failure scenarios for the different sensing modalities. Some examples of such scenarios in which different modalities fail can be seen in Figures 3a to Figure 3b. Here, dust is



(a) Visual failure with dust.



(b) Thermal and visual failure with dust and low thermal gradients.



(c) All exteroceptive sensors fail in intense dust.

Fig. 2: Motivation for Heterogeneity and Redundancy of sensors. For (a)-(c), Top left is visual, bottom left is thermal, right is a point cloud from a 360 degree LiDAR.

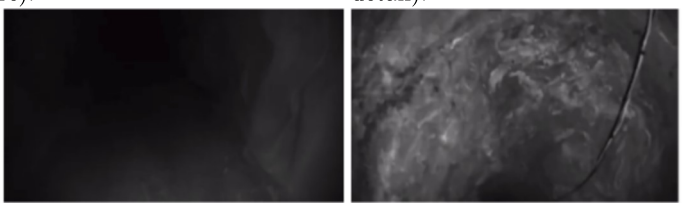
causing issues with visual sensors (Figure 3a), as does low thermal gradient for thermal cameras (Figure 3b). Intense dust can even cause issues with LiDARs (Figure 3d), requiring reliance on sensors such as an IMU.

In addition to heterogeneity, redundancy in hardware is an important factor to reduce chances of failure. For example, a drone performing fast rotations in yaw has higher chances of failure if the VIO is running on a forward facing camera compared to a downward facing camera. However, in the latter case, the VIO is susceptible to failure if the drone is performing fast rotation in pitch, moving very close to the ground at high speeds or if there is a lack of visual features on the ground. Additionally, there may be different amounts of visual texture or lighting facing forward, down or up. This can especially be the case in dark underground environments, where illumination of the scene comes from light sources on the robot (see Figure 3). Hence, using both cameras can reduce the chances of overall system failure at the cost of more mass, power and computational resources.

Finally, a robust mechanical design of the robot hardware can allow autonomy to have significantly higher tolerances to avoid failure. For example, by adding propeller guards one can reduce the safety constraint from completely avoiding obstacles to bumping at low velocities. An important consideration in this architecture is that incorporating redundancy, heterogeneity and mechanical robustness on robots such as drones, can result in significantly lower flight time due to weight and power constraints, as demonstrated in the validation section.

(a) Forward facing when near the ground (good lighting and texture).

(b) Upward facing when near the ground (poor lighting and limited detail).



(c) Forward facing when flying and in a large tunnel, which shows poor illumination of the scene.

(d) Upward facing in the same scenario as (c), showing good illumination and details of the ceiling.

Fig. 3: Contrasts in data from visual cameras pointing different directions, using onboard illumination in an underground environment.

2.2 State Estimation using Heterogeneous-Redundant Odometry - HeRO

Our objective is to estimate the following states along with their quality:

$\mathbf{p} \in \mathbb{R}^3$	Position of body frame w.r.t. world frame
$\mathbf{v} \in \mathbb{R}^3$	Velocity of body frame w.r.t. world frame represented in body frame
$\mathbf{a} \in \mathbb{R}^3$	Acceleration of body frame w.r.t. world frame
$\mathbf{R} \in SO(3)$	Orientation of body frame w.r.t. world frame
$\boldsymbol{\omega} \in \mathbb{R}^3$	Angular velocity of body frame w.r.t. world frame
$\boldsymbol{\alpha} \in \mathbb{R}^3$	Angular acceleration of body frame w.r.t. world frame
$Q_x \in \{Good, Bad\}$	Quality of state x

Note that all attributes are represented in world frame unless stated otherwise.

Further, the quality of the state is restricted to binary but can be generalized to higher resolutions and even continuous representation.

The states are used for motion planning and control, whereas the quality metrics are used by the behavior planner to select the appropriate mobility service for the current mission task (see Table 1).

Table 1: Summary of behaviours given variations in quality of state estimates

State Quality: Good Medium Bad Doesn't Matter

Case	Estimation Modality			State Quality					Mobility
No.	xIO	Height Est.	IMU	p_x, p_y, p_z	${}^g p_z$	v_x, v_y	v_z	a, R, ω, α	
1									Global
2									Local
3									Closed Loop on z
4									Local
5									Open Loop
6									Open Loop

HeRO consists of three main components: 1) Odometry Algorithms 2) Quality estimation 3) Switching logic. A set of odometry algorithms are executed on the data from the set of sensors. The quality of each odometry source is analized by performing confidence checks on the data from sensors and odometry algorithms. Finally, the switching logic uses the quality estimate to switch to the appropriate odometry source for planning and control. The switching logic also triggers re-initialization of the odometry algorithms once a failure is detected (this capability has to be incorporated if the method is a commercial off-the-shelf solution). These elements are detailed in the following.

A. Odometry Algorithms

The odometry algorithms considered are based primarily on visual, LiDAR and thermal sensors, however Radar-based and sonar-based algorithms could similarly apply. The primary motivation for the odometry algorithms is to provide state information for control. In particular, there is a requirement for the position orientation, velocity, and ideally acceleration to be estimated. A subset of odometry algorithms is independent on the IMU, such as Visual Odometry (VO) and LiDAR Odometry (LO). Such examples may include ORBSLAM2 (a VO) and LOAM (a LO). These algorithms only estimate the position and orientation instead of the full state. Therefore, the output can be loosely fused with an IMU in a filter, such as an Extended Kalman Filter (EKF) to give the required velocity estimates. Alternatively, odometry algorithms can be tightly coupled, where the IMU is integrated with how the primary sensor data is processed, and the output includes the full state. ROVIO is such an example of Visual Inertial Odometry (VIO).

While one of these algorithms can provide the required state information for control, there are particular scenarios in which each of the algorithms fails. These failure scenarios are different, though, for each modaility, as outlined in Table 2. Visual, LiDAR, and Thermal Inertial Odometry Algorithms.

Visual methods that operate on images in the visible spectrum are susceptible to failure in the presence of dust/obscurants and featureless environments. As expected, these approaches also fail in low-light environments, and one can use active lights to illuminate the environment. However, in the presence of

obscurants, illumination can intensify the obstruction (Figure 3a). Fortunately, visible-range cameras are widely available at low-cost, low-power and in small form-factors, hence can be deployed pointing in multiple different directions. Each direction can then run different VIO algorithm instances for each pointing direction. Note that this approach is in contrast to a single VIO pipeline that uses multiple cameras. Such an approach requires accurate estimation of extrinsic parameters and may not be possible with many COTS solutions. Instead, HeRO supports working with several VIO algorithms running in parallel as opposed to requiring one tightly coupled VIO.

LIO methods are robust to low-lighting, environments with a lack of visual features, and can work in a moderate amount of dust; however, they fail when the environment has a self-similar structure. For example, when a drone is moving in a long corridor with perfectly flat walls, the distance along the medial axis of the corridor becomes unobservable.

TIO addresses many of these issues, but it generally fails when there is a low thermal gradient in the environment, leading to few detectable features. Examples of this include a drone flying over a ground made of concrete with a downward facing thermal camera at night when the temperature reaches a steady-state, or deep in subterranean environments, such as tunnels, which are never exposed to sunlight as shown in Fig. 3b, 3d.

Finally, all the methods mentioned above are not robust to fast motions, especially rotations. In conclusion, there is no single solution that is robust to every failure case. Thus, HeRO runs several odometry algorithms in parallel to maximize the probability of having at least one stable state estimation source for planning and control.

Table 2: Summary of different failure modes for Inertial Odometry algorithms

xIO performance: Good Medium Bad

Algorithm	No Visual Features	Low Light	Self-similar Structure	Uniform Temperature	Dust/ Obscurants	Fast Motion
VIO						
LIO						
TIO						

B. Confidence Checks

Once we have multiple sources of odometry algorithms with disparate modes of failure, the next challenge is to identify those failures so that the optimal odometry source can be selected. Before continuing in further detail, the definition of failure in this context will be clarified. At a high level, the goal is for reliable, high-rate state estimates for control/planning and failure is anything that jeopardizes that goal. Early identification of the failure allows the system to perform actions to recover from failure or perform safety fall-backs to avoid

hard collisions. To identify the type of failures as described in Table 2, HeRO first performs confidence checks at the hardware-level by using the data from sensors. The types of checks that are done here include:

1. Rate of sensor output
2. Overall intensity of the image
3. Intensity variation in the image
4. Distance between first and last return of a LiDAR beam (generally an indicator of dust)
5. Number of invalid scan points

Next, HeRO performs confidence checks using data from the odometry algorithms, where the goal is to catch the failures depicted in Figure 4, ideally before they occur. These failures can come in a variety of forms, be it a gap in state updates, a divergence of the estimate, or rapid jumps, all of which lead to issues in control. The failure could be caused by limitations of the sensor or the overall odometry algorithm. The checks to catch these failures include:

1. Rate of algorithm output, such as pose measurements, from the front-end (catches gaps)
2. Rate of change of the position estimate (catch jumps)
3. Trace of covariance matrix on the estimates (catch divergence)
4. Algorithmic metrics on:
   (a) Number/Quality of features tracked
   (b) Number/Quality of In-lier features from scan matching/RANSAC
   (c) Rate of change of the above metrics

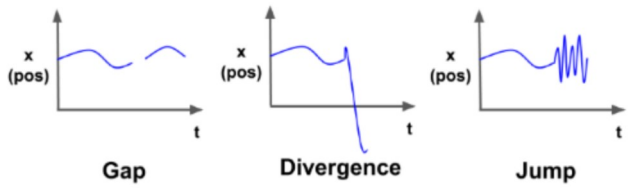


Fig. 4: Different failure modes for odometry algorithms

There is much interesting research that can be done into developing adequate confidence checks to catch failures early enough to prevent negative consequences. An implementation of a subset of these these tests is shown in the experiment section.

C. Switching Logic

Switching logic in HeRO uses the odometry estimates and confidence checks to generate a state and quality estimate that will be used by the planner and

controller. Further, when the quality of an odometry algorithm is significantly low, the switching logic also triggers a re-initialization of the failed odometry algorithms.

The simplest version of this logic can use a ranking of odometry algorithms and select the highest ranking output that has not failed. When that algorithm fails a confidence check, then the next algorithm in the ranking is selected. Another alternative that minimizes the number of switches is described along with experimental results in the next section.

More advanced logic could be developed when multiple algorithms are working to generate a state estimate by performing voting that is weighted by the quality of the algorithm. The comparison between streams also provides a third avenue for confidence checks. For instance, with three or more streams, a voting scheme can be used to identify outliers.

When switching between output state estimates, care is needed to ensure the output state estimate is consistent, with continuous pose estimates. This consistency can be achieved by iteratively composing an incremental pose from the selected odometry source to the pose from the previous state estimate.

As described in the introduction, the architecture described here focuses on a loosely coupled, multiplexing approach, as opposed to a tightly coupled, fused approach. Nonetheless, a fusion of multiple algorithms and sensors can still be incorporated into this framework, as another odometry algorithm, or by fusing all state estimate streams that pass the confidence tests.

A quality estimate is generated for each state depending on the confidence checks in the odometry source, as shown in Table 1. Note that "medium" quality of a given odometry algorithm can occur on the failure of the front-end of a VIO (e.g. loss of feature tracking). In this case, IMU measurements can be propagated to continue to get reliable velocity estimates for a short period even though position estimates may be bad. Velocity estimates alone are sufficient for control over such a period.

2.3 Planning and Control

The end goal of estimation is generally for planning and control (see Figure 1). Here we treat planning and control as having three layers: behavior planning, motion planning, and control. We discuss each layer and its adaptation to varying levels of state quality. Our choices for how we discretize our estimate of state quality is dictated by the needs of each layer within the planning and control module.

The top layer within our planning and control framework functions as a state machine, choosing what kind of behaviors the robot should execute depending on the available quality of the state estimate. Table 1 outlines the specific decisions this state machine takes within HeRO. The middle layer performs the appropriate actions decided by the behavior planner, generating desired trajectories for the robot to follow. The lowest layer, the controller, tracks desired trajectories and closes the loop directly on the provided state estimates from HeRO.

When reliable estimates of the robot's position are available, a breadth of autonomous actions which depends on position, is available for use. Examples of these actions include building occupancy-grid type maps, running path planning algorithms for reaching a desired location in the map, or reasoning about high-level, *global* goals and executing them appropriately. Many of these autonomous algorithms rely on continuous estimates of position, which HeRO provides. Without guaranteeing this continuity, many localization-dependent algorithms would fail, such as building occupancy-grid based maps using LiDAR point cloud data.

Without reliable position estimates, the robot is restricted in the set of reasonable behaviors it can take. We refer to these behaviors as being more *local* in nature since they must depend on having reliable estimates of velocity, attitude, accelerations, etc., but not position. Examples of such behaviors include wall-following, obstacle avoidance and stay-in-place (See [11]). These behaviors are often robust and locally optimal. For instance, [12] showed fast, agile flight of quadrotors using only instantaneous point-cloud information for obstacle avoidance. While effective, these behaviors often require some hand-tuning and assumptions on the topology of the environment in which the robot operates (e.g. flying through a long tunnel can be accomplished with simple wall-following behaviors).

If it is not possible to maintain high-quality estimation of velocities, then all is not lost. Specific *open loop* behaviors can be performed to reduce the likelihood of catastrophic loss of the vehicle. For example, a drone can maintain zero roll and pitch and slowly reduce its thrust to gradually land. Once landed, the chances of some odometry algorithm re-initializing dramatically increase. By not moving, camera artifacts like motion blur are reduced. Moreover, by not spinning the propellers, the effect of dust is mitigated.

Note that Table 1 differentiates between reliable estimates of velocity and position in xy vs z. Generally, drones use a height sensor to keep track of the ground, and with certain assumptions of the flatness of the terrain, can use this height sensor in a loosely coupled estimation algorithm to estimate the relative height and velocity in z of the drone. With this setup, landing can be safer and more controlled, which is an example of behaviors we denote as *Closed Loop in* z.

Given the state quality and estimate, our planning and control architecture intelligently switches between behaviors, where each behavior relies on having good quality estimates of subsets of the state. The planner decides which motion planner to use (global or local), and what type of control to use (cascaded, full PID, or open loop). Some odometry algorithms give mixed results on their state estimation quality - for example, having reliable velocity estimates but an intermittent loss of good position estimates. Here we take the approach to use as much state information as possible while checking its reliability and accuracy.

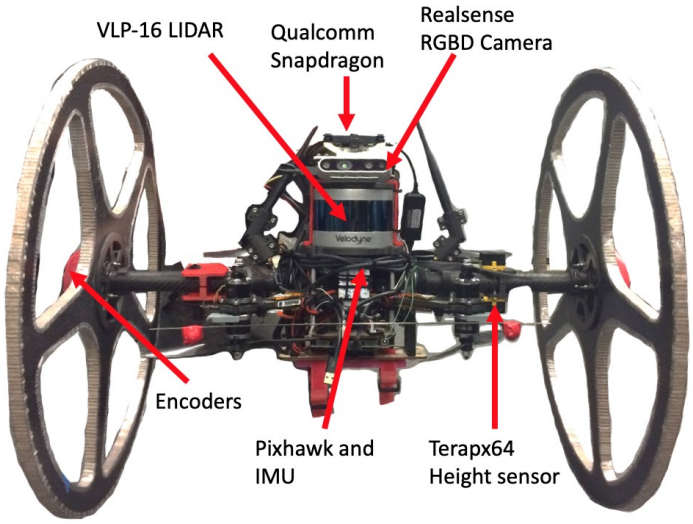


Fig. 5: The *Roll-o-copter*: a hybrid ground/aerial robot

3 Validation and experiments

In this section, we show the validity of our state estimation framework with a specific implementation. We set up the resiliency logic supervising three odometry algorithms using vision, a LiDAR and an IMU as sensing modalities: a loosely coupled VIO with stereo cameras facing forward, a tightly coupled VIO with a monocular camera facing up, and a LiDAR-inertial odometry algorithm with a 360 degree, 16 channel LiDAR. This setup provides redundancy of sensors and heterogeneity in the algorithmic solution.

In the following we describe in detail the hardware, odometry algorithms, confidence checks and multiplexing approaches used. Moreover we include experimental results obtained from the STIX event of the DARPA subterranean challenge.

3.1 Hardware

We make use of the *Roll-o-copter*: a hybrid ground and aerial vehicle designed to both fly and roll on the ground (see Figure 5). The Roll-o-copter is a multi-rotor aerial vehicle with two passive wheels attached to each side of the frame. Although this robot is designed to take advantage of both aerial and ground terrains, in this work we focus on the aerial mobility of the vehicle and use it as flying-only platform.

In flight, *Roll-o-copter* behaves like a normal multi-rotor, with a standard set of electronic speed controllers (ESCs), motors and propellers. It possesses a

Pixhawk[1] v2.1 as flight controller, as well as an on-board Intel NUC[2] i7 Core computer. In addition to the standard multi-rotor hardware, we equipped the robot with the following sensors:

- **RealSense[3] RGBD camera.** Composed by a stereo pair of infrared (IR) cameras, an RGB camera and a structured light IR projector. This sensor has its own on-board processing which provides RGB-D depth maps, point clouds, color, and IR images.
- **Velodyne 360° VLP-16[4] LiDAR.** Rotary head with 16 LiDAR rangers, providing a point cloud with a 360° azimuth angle and ±15° elevation angle field of view.
- **TeraRanger evo 64px[5] IR time-of-flight range sensor.** We use 2 rangers (upward and downward facing), providing top and bottom clearances for collision avoidance.
- **Pixhawk v2.1 flight controller.** Composed by an on-board IMU (3-axis gyroscopes and accelerometers) and processing, including a Kalman filter for its own state estimation.
- **Qualcomm Snapdragon[6].** A self-contained flight controller accompanied by an IMU (3-axis gyroscopes and accelerometers) and two monocular cameras (forward and upwards facing) for tightly coupled VIO.

3.2 HeRO stack

We validate the HeRO approach with two VIOs and one LIO in the stack of methods, which provide the required capabilities to fly in complex and perception-challenging environments. These methods are running in parallel at different frequencies and using different sensor sources, as detailed in the following.

- **Infrared Stereo Visual Inertial Odometry.** We take advantage of ORB-SLAM2 (OS2) [13] running using images from the IR stereo camera (RealSense RGBD). This approach produces 6D pose estimates (3D translations and 3D rotations) and can run up to 60Hz. These estimations are fused with IMU data (running at a frequency of around 1kHz) within the Pixhawk flight controller. This approach is based on an Extended Kalman Filter [14] running on a delayed time horizon and a complementary filter producing the inertial solution at the non-delayed time horizon.
  Using a stereo odometry algorithm enables us to re-initialize it in flight without the dependence of parallax movements such as in monocular VIO methods. To keep the filter consistent we use a transform manager that

[1] http://www.pixhawk.org
[2] https://www.intel.com/content/www/us/en/products/boards-kits/nuc/boards/nuc7i5dnbe.html
[3] https://software.intel.com/en-us/realsense/d400
[4] https://velodyneLiDAR.com/vlp-16.html
[5] https://www.terabee.com/shop/3d-tof-cameras/teraranger-evo-64px
[6] https://developer.qualcomm.com/hardware/qualcomm-flight-pro

adapts the input signal to coherent values in the Mahalanobis distance sense. Moreover, we are controlling using velocities and using a re-planning strategy running at 20Hz, hence we choose robustness over accuracy and disable the loop-closure features of OS2 (the drift in position is not affecting the filter velocity estimates).

- **Monocular Visual Inertial Odometry.** We use the Qualcomm's Snapdragon Flight (QSF) platform running their commercial off-the-shelf VIO from mvSDK as the second source of odometry. As in the case of OS2 odometry, here we also incorporated some modifications to allow re-initialization during flight. The incorporation of this approach also demonstrates the capability of our framework to easily integrate closed-source commercial VIO solutions. The state estimation runs at 25Hz.
- **LiDAR Inertial Odometry.** The third source of odometry consists on fusing 6D pose estimates from a LiDAR odometry (LO) approach with IMU data within a regular EKF scheme. The pose estimates from the LO are produced at 20Hz and the IMU runs at 200Hz. We use the same IMU as for the IR Stereo VIO but this time with the data externalized from the Pixhawk flight controller at 200Hz.

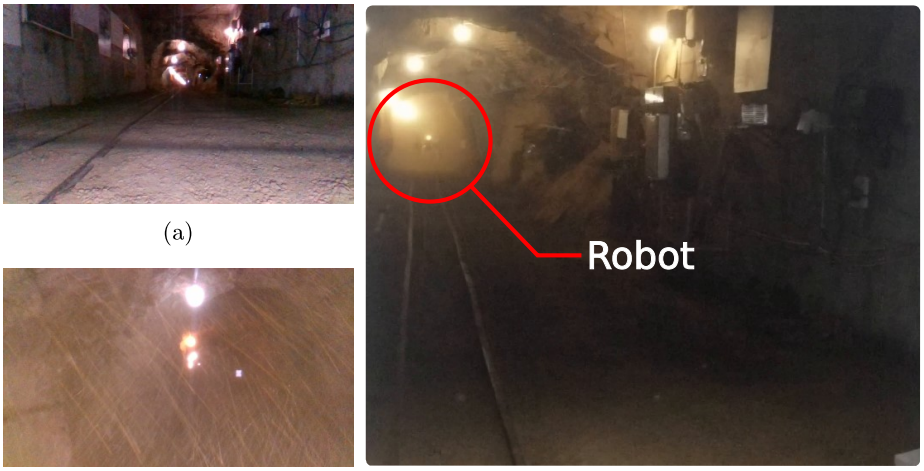
Notice how the above choices allow us to use three different sensing modalities (IR stereo forward-facing, monochrome monocular upward-facing and LiDAR) with three different estimation algorithms. All sensors and approaches are prone to different possible failures, minimizing the occasions where they all fail simultaneously, thus achieving the required level of robustness, as will be shown in the following sections.

3.3 Resiliency logic

In these experiments we use the following confidence tests, set by observing the types of failures the methods are prone to.

- **Frequency:** Most common mode of failures of OS2 occurs due to feature tracking failures when the drone is executing a fast motion or due to presence of featureless environments. In this case, the frequency of the measurement updates goes down while failing. This policy helps catching "data gaps".
- **Estimation covariance:** The uncertainty of the estimated pose from QSF significantly increases during failure. Hence, we detect these failures by setting a threshold on the trace of the estimation covariance matrix (experimentally determined). This policy helps catching "data divergence".
- **Sudden position changes:** If the estimation method results inconsistent, it might still produce an output although the covariance of the estimations might not reflect it. To detect these failures we set a confidence check on sudden position changes to catch "data jumps".

In order to use the best of the inputs (*i.e.*, best estimation) we choose for these experiments a multiplexing strategy inside the resiliency logic. The procedure



(a) (b) (c)

Fig. 6: Images from the robot frontal camera (a and b) and external view (c) during a real experiment.

consists on running the above mentioned confidence checks on the estimations. If an estimation is nor working or degraded we trigger the re-initialization of its estimation engine and, if necessary, switch to a different estimation source. In order to avoid possible jumps in the output we guarantee the continuity by concatenating the relative motions between estimations. Notice that a fusion of the estimates with traditional techniques is not possible here as there will arise correlations between them that are difficult to be modelled.

3.4 Navigation in a mine, example of usage in the STIX event of DARPA subterranean Challenge

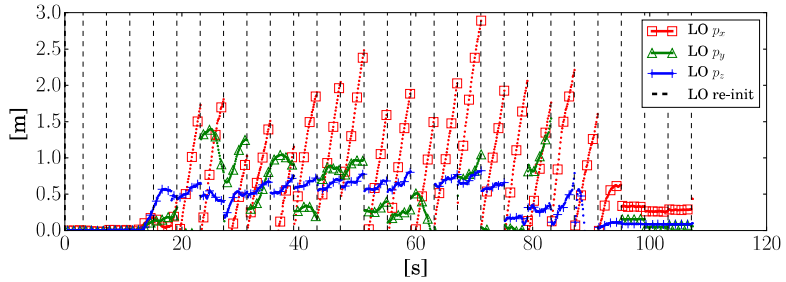
The experimental scenario is part of the DARPA subterranean challenge[7] and, specifically, we show results from the official integration event (STIX). In these experiments we deploy the *Roll-o-copter* in the entrance of a gold mine and set an autonomous exploration mission. Figure 6 shows an example of one of the experimental runs, and shows how challenging is the environment in terms of perception, clearly justifying the use of a resilient state estimation approach (*e.g.*, notice the dusty environment comparing Figure 6 a) where the robot did not start and b) and c) while flying). In these experimental runs of the main STIX event we were able to accomplish our missions with *Roll-o-copter*, exploring two different entrances of the mine and validate our resilient state estimation strategy.

[7] https://www.subtchallenge.com/

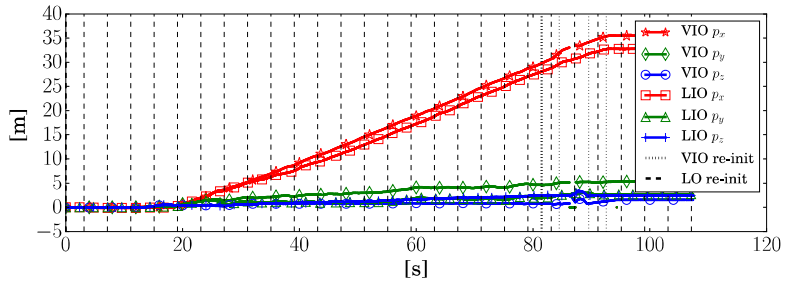
An example of these runs (mine entrance 1) is shown in Figure 7. Here we show the different types of re-initialization, depending on which element is re-initialized. A first case consists of partly re-initializing a loosely-coupled approach, for example resetting one of the inputs of the fusion algorithm. In this case, we cannot detect discontinuities at the resiliency logic level because the fusion still produces outputs from the other sensor modalities; however, we can still detect divergences of covariance which would indicate not to use its estimate. In order to avoid inconsistencies in the fusion algorithms, whenever we re-initialize a particular method that is used in a fusion later on, we keep track of pose discontinuities with a transform manager. Figure 7 a) shows this behaviour, and how re-initializing manually every 5s a LiDAR odometry (LO) provokes LO position estimate jumps. Thanks to the transform manager, the fusion of this LO with an IMU (LIO) is kept consistent and continuous (see Figure 7 b) where we show the LIO output and an OS2 VIO for comparison purposes). Figure 7 c) shows an example of resulting x-body axis velocities of all methods in the HeRO stack, running two VIOs (VIO1 is OS2 and VIO2 is QSF) and the LIO test for the same experiment. In this run the resiliency logic was directly selecting VIO1 (used for autonomous navigation) and it requested several times a re-initialization of the VIO2 due to failures.

The modularity of the presented state estimation framework allow us to run different methods on the HeRO stack. For example, in the case of a drone where the available payload does not allow carrying a 3D LiDAR, we can still use redundant VIOs. An example of this case is shown in Figure 8 corresponding to a different experiment in the entrance 2 of the mine. In these figures we show several useful information that validates the use of the presented approach, specifically:

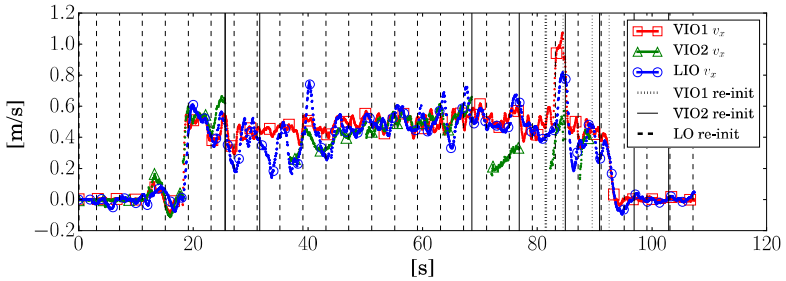
- **VIO estimations:** we show the estimated position (first three plots) and velocities (latter three plots) of OS2 (VIO1 in red with squares) and QSF (VIO2 in green with triangles), together with the respective output of the resiliency logic (blue solid line). Notice how the resilient output does not always overlap with the estimations because it keeps continuity during switches.
- **Channel selection:** On all Figure 8 we overlap the channel selected by the resiliency logic (magenta and specification on the right axis) while switching between methods.
- **Re-initialization triggers:** When a method is not passing the confidence checks, it is required to re-initialize. These triggers are shown in all images as vertical lines.
- **Mobility services:** This is the main motivation for this work. Depending on the available estimations (full or partial state) we can make use of different mobility services. The available mobility services are shown in all plots with coloured areas. When there is at least one method providing the full state (*i.e.*, position and velocity estimates) we can run global mobility services (*e.g.*, trajectory or global target tracking). In those cases where there is no full state available because we are fully re-initializing a method (*i.e.*, the method and its fusion engine), we cannot use position estimates for the



(a) LiDAR odometry (LO) X-Y-Z positions with LO re-initialization triggers every 5s.



(b) VIO (OS2) and LIO X-Y-Z position estimates with LO re-initialization triggers every 5s.



(c) VIO (OS2) and LIO X velocity estimates with LO re-initialization triggers every 5s.

Fig. 7: Results from a real experiment with the *Roll-o-copter* flying autonomously in a tunnel (see Figure 6).

mobility services but velocity estimates are usually still available, hence we can use local mobility services (*e.g.*, wall following strategy controlling using velocities). In case of not being able to re-initialize the estimation methods, we fly on dead-reckoning so we can only use attitude mobility services. In particular for this experiment, for safety reasons we are land after 3s of dead-reckoning flight. This strategy helps us to let the dust settle down and